Buffer Overflow

Exploitation

A program is a set of rules that tell a computer exactly what to do: letter of the law principle

Exploitation of a program is a clever way of getting the computer to do what you want it to do, even if that was not what it was designed for

Security holes are oversights or flaws in the design of program...

Unintentional oversights...

What's written in a program doesn't always coincide with what the programmer intended to do

and this can have some pretty catastrophic repercussions

Off-by-one

An error where the programmer has miscounted by one

Q: if you're building a 100 foot fence, with fence posts spaced 10 feet apart, how many fence posts do you need?

A: The obvious answer is 10, but it is actually 11. *Fencepost error* is when you accidentally count the spaces instead of items.

Off-by-one: ranges

Imagine you are processing a range of items from N to M

Q: If N = 8 and M = 17, how many items are you processing?

A: The obvious answer is 17 - 8 = 9, but it is actually 17 - 8 + 1 = 10

It seems counterintuitive - because it is, and that’s why these errors happen

Fencepost errors goes unnoticed...

...because programs aren't tested for every single possibility

but, when the program is fed the input that makes the effects of the error manifest, there is an avalanche effect on the rest of the program logic

this means that when properly exploited, an off-by-one error can cause a seemingly secure program to become a security vulnerability

OpenSSH

Meant to be a secure terminal communication program suite to replace unencrypted services like telnet

but there was a off-by-one error that was heavily exploited

```
if (id < 0 || id > channels_alloc) {
```

```
if (id < 0 || id >= channels_alloc) {
```

What breeds exploitable code?

A program is modified quickly without being tested thoroughly

Microsoft's IIS webserver serves web content to users: it needs to allow users to read, write, execute programs in certain directories and only those directories

The program has *path-checking* code to prevent users from using backslash to go backward through the directory tree and enter other directories

However, the support of Unicode meant that there were multiple representations of of the backlash character (such as %5c) but translation was done AFTER path-checking

Legal loopholes

A 21-year old hacker, David LaMacchia, set up Cynosure for the purposes of software piracy

Software companies claimed they lost millions of dollars

This wasn’t criminal conduct under the Copyright Aact since the infringement was not for the purpose of commercial advantage

Lawmakers never anticipated that someone would conduct piracy without a motive for personal financial gain

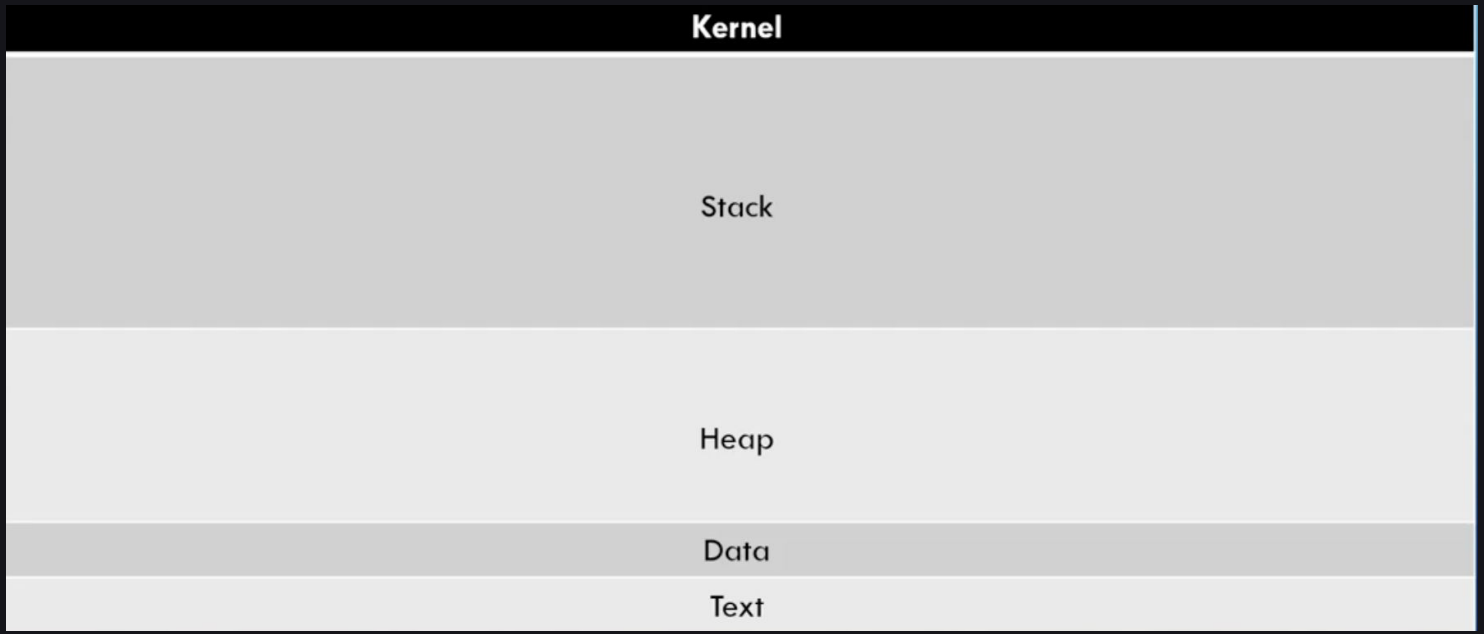
Generalized Exploit Techniques

The same time of mistake is used in many different places, so generalized exploit techniques have evolved to take advantage of these mistakes

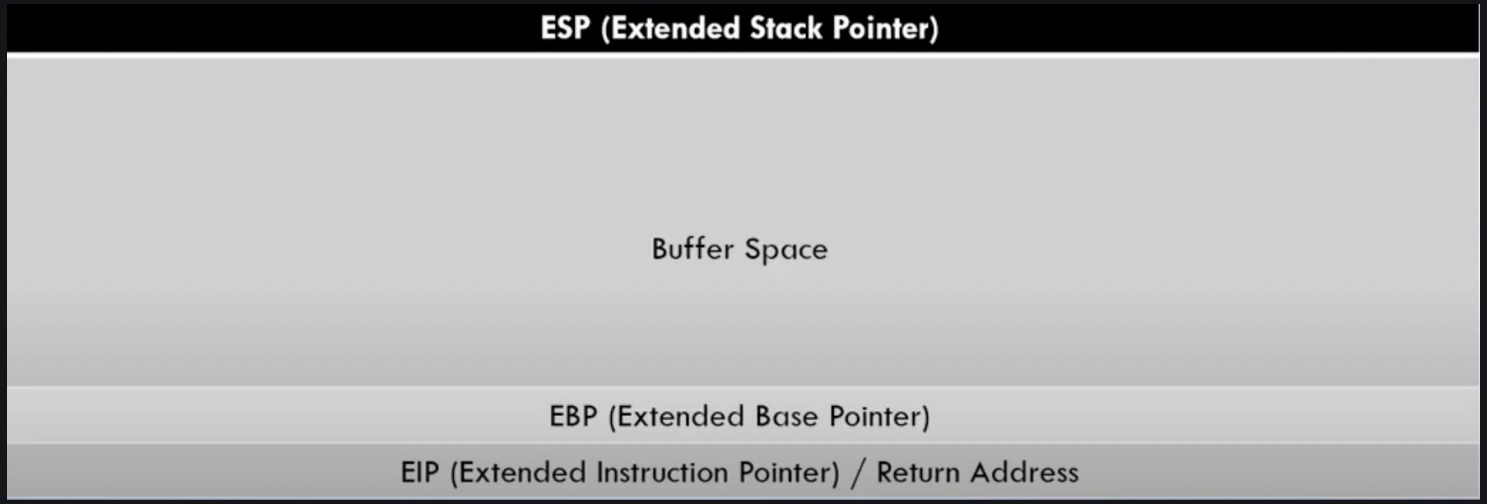
Most of these exploits have to do with **memory corruption,** such as buffer overflows

the ultimate goal: take control of target program's execution flow and run malicious smuggled shell code, *execution of arbitrary code*

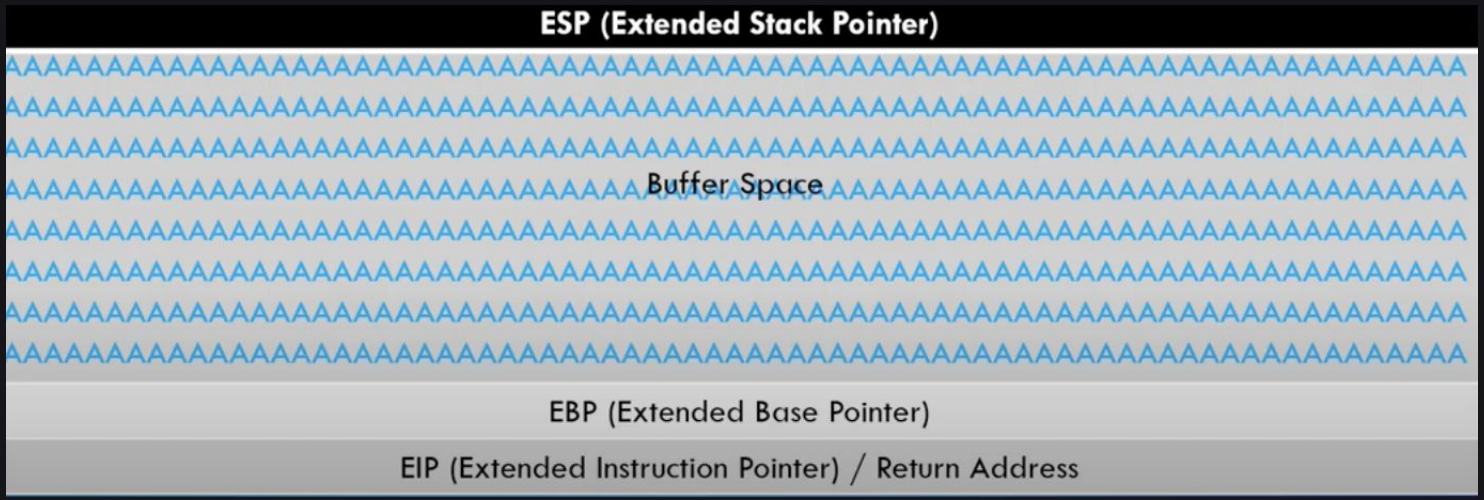
Memory segments used by a program



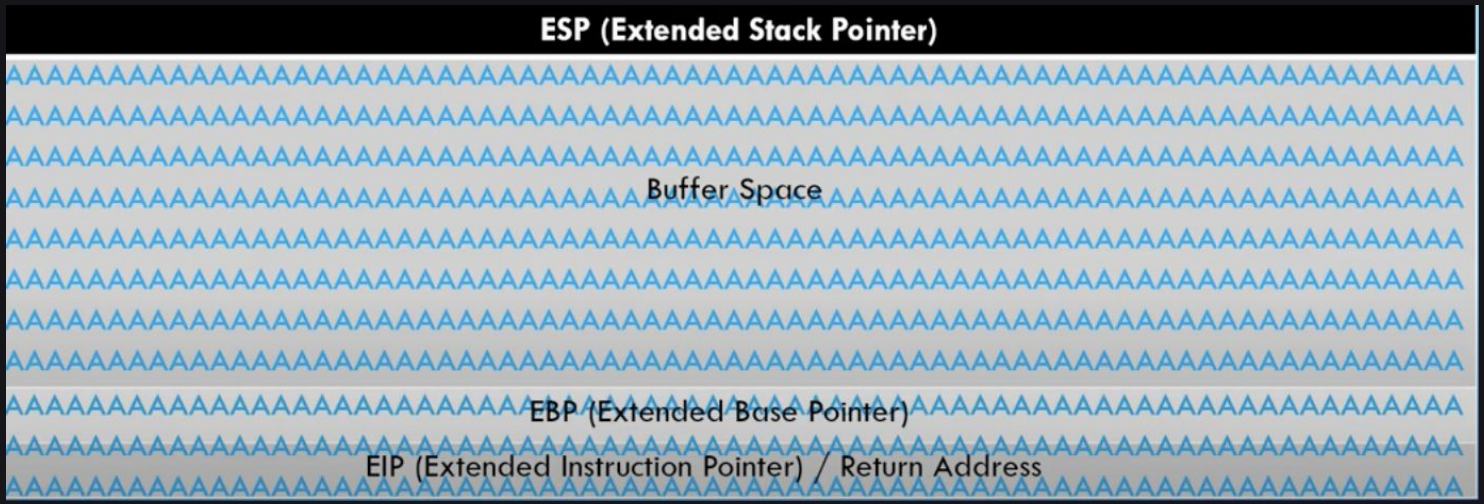
Stack


ESP (Extended Stack Pointer)
Buffer Space
EBP (Extended Base Pointer)
EIP (Extended Instruction Pointer) / Return Address

Stack


ESP (Extended Stack Pointer)
Buffer Space
EBP (Extended Base Pointer)
EIP (Extended Instruction Pointer) / Return Address

Stack


ESP (Extended Stack Pointer)
Buffer Space
EBP (Extended Base Pointer)
EIP (Extended Instruction Pointer) / Return Address

Buffer Overflows

C is a high-level programming language that assumes that the programmer will be responsible for data integrity

What happens if the responsibility shifted to compiler? Integrity checks on every variable

C's simplicity means more control, but a higher chance of **buffer overflows and memory leaks** if the programmer isn't careful

if a variable is allocated memory, there are no safe guards to ensure that the contents fit in that allocated memory

Overflow example

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
   int value = 5;
   char buffer_one[8], buffer_two[8];

   strcpy(buffer_one, "one"); /* Put "one" into buffer_one. */
   strcpy(buffer_two, "two"); /* Put "two" into buffer_two. */

   printf("[BEFORE] buffer_two is at %p and contains \'%s\'\n", buffer_two, buffer_two);
   printf("[BEFORE] buffer_one is at %p and contains \'%s\'\n", buffer_one, buffer_one);
   printf("[BEFORE] value is at %p and is %d (0x%08x)\n", &value, value, value);

   printf("\n[STRCPY] copying %d bytes into buffer_two\n\n",  strlen(argv[1]));
   strcpy(buffer_two, argv[1]); /* Copy first argument into buffer_two. */

   printf("[AFTER] buffer_two is at %p and contains \'%s\'\n", buffer_two, buffer_two);
   printf("[AFTER] buffer_one is at %p and contains \'%s\'\n", buffer_one, buffer_one);
   printf("[AFTER] value is at %p and is %d (0x%08x)\n", &value, value, value);
}
```

Overflow example

```
reader@hacking:~/booksrc $ gcc -o overflow_example overflow_example.c
reader@hacking:~/booksrc $ ./overflow_example 1234567890
[BEFORE] buffer_two is at 0xbffff7f0 and contains 'two'
[BEFORE] buffer_one is at 0xbffff7f8 and contains 'one'
[BEFORE] value is at 0xbffff804 and is 5 (0x00000005)

[STRCPY] copying 10 bytes into buffer_two

[AFTER] buffer_two is at 0xbffff7f0 and contains '1234567890'
[AFTER] buffer_one is at 0xbffff7f8 and contains '90'
[AFTER] value is at 0xbffff804 and is 5 (0x00000005)
reader@hacking:~/booksrc $
```

Overflow example

```
reader@hacking:~/booksrc $ ./overflow_example AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[BEFORE] buffer_two is at 0xbffff7e0 and contains 'two'
[BEFORE] buffer_one is at 0xbffff7e8 and contains 'one'
[BEFORE] value is at 0xbffff7f4 and is 5 (0x00000005)

[STRCPY] copying 29 bytes into buffer_two

[AFTER] buffer_two is at 0xbffff7e0 and contains
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
[AFTER] buffer_one is at 0xbffff7e8 and contains 'AAAAAAAAAAAAAAAAAAAA'
[AFTER] value is at 0xbffff7f4 and is 1094795585 (0x41414141)
Segmentation fault (core dumped)
reader@hacking:~/booksrc $
```

THANK YOU!
https://www.umisc.info/join/
code credits: the art of exploitation